

Deployment modeling in Rational Software Architect Version 7.5, Part 1: Anatomy of a topology model

Skill Level: Intermediate

[Narinder Makin \(nmakin@us.ibm.com\)](mailto:nmakin@us.ibm.com)
Software Architect
IBM

02 Dec 2008

IBM® Rational® Software Architect Version 7.5 contains a powerful Deployment Architecture Platform to design and understand IT systems and their complex relationships visually. This platform is built on an extensible, strongly-typed model which provides a framework for describing any IT system. This article series presents the anatomy of the topology model that is the heart of this feature. These topology models describe the logical abstractions (or the concrete definition) of the building blocks of any IT system. They also allow you to express different kinds of relationships between those system elements. The basic concepts of the topology model (including the fundamentals of topology, units, requirements, and capabilities, along with relationships like dependency, hosting, and members) are discussed in this part of the series. It also provides an example of how the model can be used to describe a particular IT system, integrating information about a particular domain with the predefined types in the model.

Introduction

IBM® Rational® Software Architect Standard Edition Version 7.5 and IBM® Rational® Software Architect for WebSphere Software Version 7.5 (Rational Software Architect V7.5, for short) contain a powerful deployment architecture platform to model IT systems and their complex relationships visually. It is built on an extensible, strong, typed model called a *topology model*. This feature addresses the following need.

You may often be required to express the design of your IT systems in order to provide communications across departmental boundaries (for example, between development and operations). Conventional methods of transferring such information include notes, e-mails, spreadsheets, slides, or Microsoft® Visio diagrams. There has always been a need for a formal method to communicate this design, and a system to validate that communication. The absence of such a solution leads to a variety of configuration issues that are seen throughout the application lifecycle, with unchecked deployments made in the target environments. This gap even worsens when a solution moves between different stages of the application lifecycle, like development, staging, testing, or operations. As a result you may have to re-architect the system, or it may be nearly impossible to replicate a successful system installation. The patterns or practices of building or deploying such systems are often documented on paper, with no way to realize or enforce them.

Typically, such IT systems require different relationships to be expressed between the pieces within the system. This is done by naming these relationships, which may vary between the different groups in an IT organization, or across organizations. Everyone understands that, more or less, there are some common semantics behind these relationships, but they lack the means of commonly expressing these. The topology model not only provides the means to express these relationships in a common language, but further classifies these relationships based on their semantics (like dependency, hosting, or member). This allows you to define the relationships in a controlled fashion that you can read, but that is also interpretable by software tools to produce validated and resolved topologies.

The following sections discuss the basic concepts of the topology model, along with specifics about how to apply a domain knowledge to the types in the model.

Topology model concepts

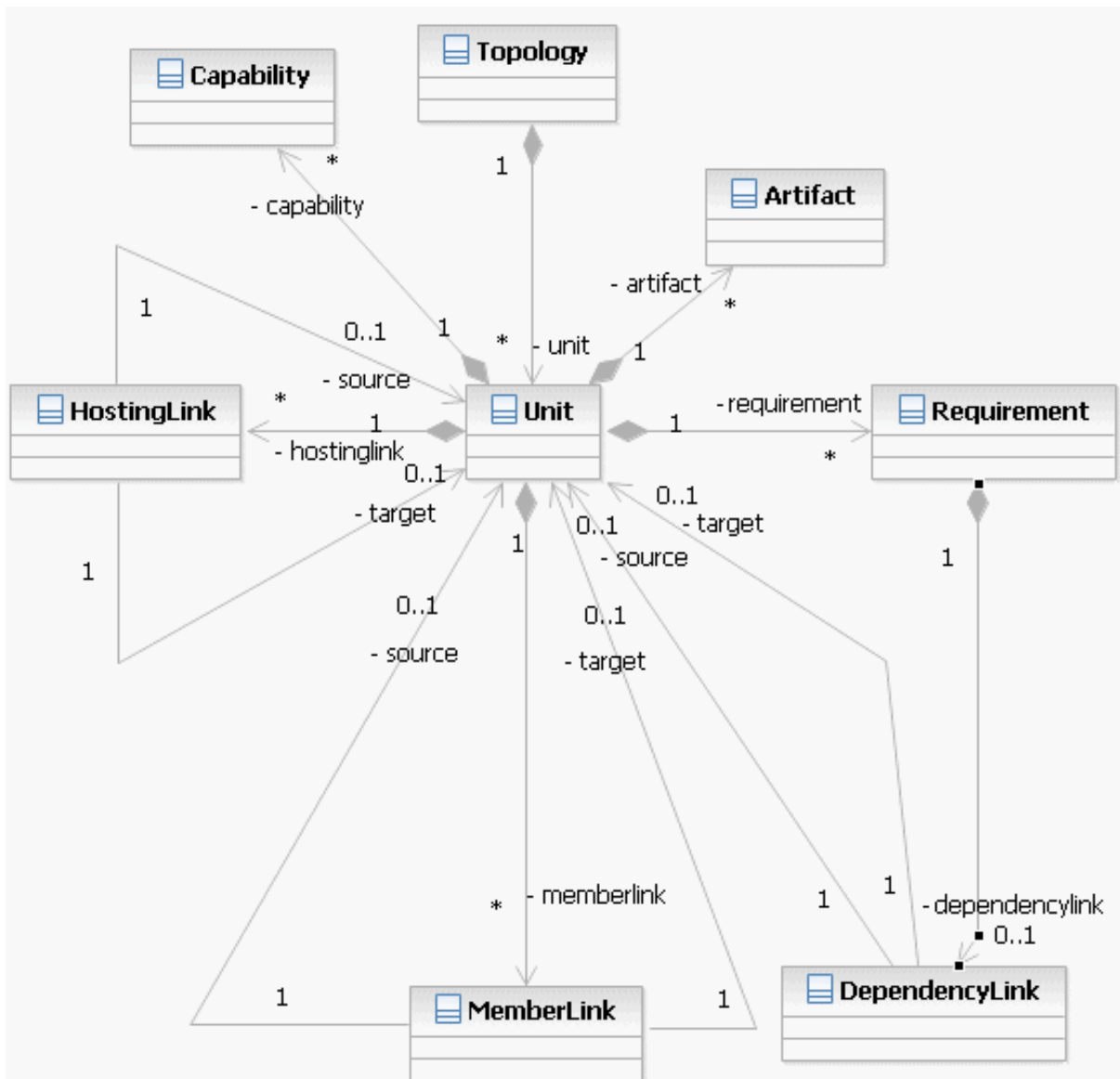
You can use a topology model to describe an IT system, or pieces of an IT system. It provides a common language to express the system. It also gives you the ability to contribute new types (or instances of new types) with respect to the domain in one or more topology models. Each of these instances in the topology model are loosely coupled with the capabilities, and can specify their requirements. The following basic concepts of the topology model are covered in this article:

- **Topology** is a type that is the top-level container of the topology model.
- **Unit** is a type that represents the unit of deployment.
- **Capability** is a type that describes an ability in the target environment.
- **Requirement** is a type that describes the need in the target environment.

- **Artifact** is a type that describes a deployable resource or an object.
- **Relationship Links**
 - **Dependency Link** is a link type that represents the binding of the functional need to the ability that satisfies it.
 - **Hosting Link** is a link type that represents the binding of the hosting need to the hosting ability that satisfies it.
 - **Member Link** is a link type that represents the binding of the containment need to the containment ability that satisfies it.

A topology model is the container of all of the units and their relationships, where units are loosely associated with the requirements and the capabilities. The following sections include a detailed description and usage of these types with respect to a sample topology model that you build during the course of this article. A basic UML representation of these types is shown in Figure 1.

Figure 1. Basic topology UML model



Topology

Topology is the top-most element of any topology model. It provides a representation of the topology model with all of the contents of the topology, including units and their relationships. A topology expressing an IT system may be as simple as describing a component of the system. It could also be complex enough to specify many components to their nth level, along with their relationships. It is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema, as shown in Listing 1. topology is uniquely identified by its name and namespace. You can also specify a display name that is consumable by the visual tools, along with a description.

Listing 1. Core namespace in core.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
description="This is a sample topology describing the anatomy of the topology model"
displayName="Sample Topology"
name="SampleTopology"
namespace="developerworks.deploy">
<!-- all other containments of the topology go here -->
</core:topology>
```

The scope of topology is not limited to describing a complete IT system, or even a major piece of it. Depending on your needs, you may dissect or collapse the contents of your environment into single or multiple topologies that can be referenced. In this case, the topologies serve as the reuse method, and can be composed to form larger topologies.

Additionally, you may also tag the topology using the `decoratorSemantic` attribute to describe its intention during the modeling. The common tags like `analysis`, `infrastructure`, and `deployment` have been predefined in IBM® Rational® Software Architect for use. In the example in Listing 2, the topology is tagged with `com.ibm.ccl.soa.deploy.core.dds`, which is interpreted by Rational Software Architect as tagged for `analysis`.

Listing 2. The decoratorSemantic attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
description="This is a sample topology describing the anatomy of the topology model"
displayName="Sample Topology"
name="SampleTopology"
namespace="developerworks.deploy"

decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads">
<!-- all other containments of the topology go here -->
</core:topology>
```

Unit

Unit is a base type that represents a unit of deployment in a topology model. Generally, a unit represents an isolated piece of an IT system that defines a set of

requirements. These requirements are to be fulfilled during deployment, and they describe a set of capabilities that the system provides. Unit type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. It is uniquely identified by its name within the scope of a topology. You can also specify a display name that is consumable by the visual tools, along with a description as shown in Listing 3.

Listing 3. Unit type

```
<core:unit
  displayName="SampleUnit"
  name="sampleUnit"
  description="This is a sample unit">
</core:unit>
```

Configuration

During deployment, there are certain pieces that are just configurations. The model allows you to set a `configurationUnit` attribute on the unit to specify that it is a configuration, as shown in Listing 4. As an example, a server can be treated as a non-configuration unit, while a datasource that is configured on that server can be treated as a configuration unit.

Listing 4. Configurations

```
<core:unit
  displayName="SampleConfigurationUnit"
  name="sampleConfigurationUnit"
  description="This is a sample configuration unit"
  configurationUnit="true">
</core:unit>
```

Conceptual

There are also times when you do not fully know the definition of the unit at the time of analysis. Thus the topology model, which represents the analytical state of the IT system or its piece, may not be able to express all of the capabilities or requirements (or even the members) of that unit. For these purposes of *intended* unit representation, an optional `conceptual` attribute with a default value of `false` exists on a unit, as shown in Listing 5. This may also be thought of as a *slot* in the topology that must be selected at a later point in time.

Listing 5. The conceptual attribute

```
<core:unit
  displayName="SampleConceptualUnit"
  name="sampleConceptualUnit"
  description="This is a sample conceptual unit"

  conceptual="true">

</core:unit>
```

Stereotype

You can further classify a unit using stereotypes as an additional keyword on the unit, as shown in Listing 6. You can then use these keywords for variety of reasons to associate behavior based on their profiles. These stereotypes also provide a means of interoperability between the stereotypes contributed by different profiles, defined on the components in the Unified Modeling Language (UML).

Listing 6. Stereotype keywords

```
<core:unit
  displayName="ServiceComponentUnit"
  name="serviceComponentUnit"
  description="This is a stereotyped service component unit">

  <core:stereotype

    keyword="service"

    profile="serviceProfile"/>

</core:unit>
```

Installation

Optionally, a unit may also specify its install state for the purposes of deployment as `unknown`, `installed`, or `not_installed`. The install state is further broken down to specify the initial install state or the goal install state of the unit with default values of `unknown`. The combination of initial and goal install state on the unit provide additional information on the deployment aspects of the unit to the publisher of the topology model. A scenario in which the initial install state may be `installed` while the goal install state is `not_installed` provides the interpretation to uninstall the unit when the topology model is published. You can also specify a `publishIntent` attribute, as shown in Listing 7, with a value of `unknown`, `publish`, or `do_not_publish` (the default is `publish`). This gives an additional hint to the publisher about whether or not to ignore the unit during publishing. This also enables incremental or isolated publishing of the topology model during the different stages of development and deployment.

Listing 7. Install state and publish intent

```
<core:unit
  displayName="SampleConceptualUnit"
  name="sampleConceptualUnit"
  description="This is a sample conceptual unit"
  conceptual="true"

  initInstallState="installed"

  goalInstallState="not_installed"

  publishIntent="do_not_publish">
</core:unit>
```

Composition

A topology may be defined to contain just one unit, or it may contain more, depending on the intention of the topology. You can define a set of topologies to create a catalog of units that are present in the IT system. These topologies act as reusable building blocks for other topologies to be built, and may contain a minimal number of isolated units. A topology can also be defined to represent the complete IT system: that topology may contain a large number of units representing each of the pieces in that IT system, as shown in Listing 8.

Listing 8. Code with several units

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
  xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
  description="This is a sample topology describing the
  anatomy of the topology model"
  displayName="Sample Topology"
  name="SampleTopology"
  decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
  namespace="developerworks.deploy"

  <core:unit

  description="This is a sample unit"

  displayName="SampleUnit"

  name="sampleUnit"/>

  <core:unit

  description="This is a sample configuration unit"

  displayName="SampleConfigurationUnit"

  name="sampleConfigurationUnit"

  configurationUnit="true"/>

  <core:unit

  description="This is a sample conceptual unit"

  displayName="SampleConceptualUnit"
```

```

name="sampleConceptualUnit"
conceptual="true"
goalInstallState="not_installed"
initInstallState="installed"
publishIntent="do_not_publish"/>

<core:unit
description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit"
name="serviceComponentUnit">
<core:stereotype
keyword="service"
profile="serviceProfile"/>
</core:unit>

</core:topology>

```

Domain Usage

Units are typically specialized for a domain, and a topology may contain units from one or more of these domains. For example, a Java™ 2 Platform, Enterprise Edition (J2EE) domain may define a J2EE Server type (`j2eeServer`) and a J2EE application component (`component.ear`) type as an extension of a core unit type. In the topology shown in Listing 9, core unit types and unit types from a J2EE domain are used together. Also note that the install state of the server unit is specified such that it remains unchanged for initial and goal install state. On the other hand, the install state on the J2EE application component specifies that the unit is not installed and needs to be installed.

Listing 9. Units from different domains

```

<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"

xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"

description="This is a sample topology describing the anatomy of the topology model"
displayName="Sample Topology" name="SampleTopology"
decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
namespace="developerworks.deploy">
<core:unit description="This is a sample unit"
displayName="SampleUnit" name="sampleUnit"/>
<core:unit description="This is a sample configuration unit"
displayName="SampleConfigurationUnit" name="sampleConfigurationUnit"
configurationUnit="true"/>

```

```
<core:unit description="This is a sample conceptual unit"
displayName="SampleConceptualUnit" name="sampleConceptualUnit"
conceptual="true" goalInstallState="not_installed"
initInstallState="installed" publishIntent="do_not_publish"/>
<core:unit description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit" name="serviceComponentUnit">
<core:stereotype keyword="service" profile="serviceProfile"/>
</core:unit>

<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
name="j2eeServerUnit"
conceptual="true"
goalInstallState="installed"
initInstallState="installed"/>

<j2ee:component.ear
description="This is an EAR Component "
displayName="EAR Component "
name="earComponent"
configurationUnit="false"
goalInstallState="installed"
initInstallState="not_installed"/>

</core:topology>
```

Capability

Capability represents the ability that a unit may provide to other units, and it carries the configuration content of the unit. Capability type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. Capability is contained in the unit and is uniquely identified by its name with in the scope of a unit. You can also specify a display name that is consumable by the visual tools along with a description. The `linkType` attribute of the capability defines the type of links a capability can be used with, as shown in Listing 10.

Listing 10. Capability

```
<core:capability
displayName="SampleCapability"
```

```
name="sampleCapability"  
description="This is a sample capability"  
linkType="any" />
```

The two broader sets of capabilities that are usually defined in a topology model fall under `dependency` and `hosting` capabilities. A `hosting` capability defines the ability for a unit to host or provide middleware services (for example, application server) to one or more other units. A `dependency` capability defines the ability for a unit (for example, datasource) to do work for its peer units, which may be co-hosted or hosted on different units. The model does not define any formal types for the above two categories of capability, and instead uses the `linkType` attribute to be specified as `hosting` or `dependency`. Typically, `dependency` capabilities have the `linkType` as `dependency`, while the `hosting` capabilities define the `linkType` as `any` to express the fact that even though they are hosting, others may have indirect dependencies on their configuration.

A hosting capability can be defined as shown in Listing 11.

Listing 11. Hosting capability

```
<core:capability  
  displayName="SampleHostingCapability"  
  name="sampleHostingCapability"  
  description="This is a sample hosting capability"  
  linkType="hosting" />
```

A dependency capability can be defined as shown in Listing 12.

Listing 12. Dependency capability

```
<core:capability  
  displayName="SampleDependencyCapability"  
  name="sampleDependencyCapability"  
  description="This is a sample dependency capability"  
  linkType="dependency" />
```

Capabilities are associated with a unit in the topology model to represent the abilities of the unit, but the schema for the topology model does not enforce any strict association between unit and capability types. These are loosely coupled together in the topology models to provide utmost flexibility between the unit and the available capability types. For instance, perhaps the same Capability type can be associated

with one or more unit type on ad hoc basis.

A dependency capability can be associated to a configuration unit as shown in Listing 13.

Listing 13. Associate a capability with a configuration unit

```
<core:unit
  displayName="SampleConfigurationUnit"
  name="sampleConfigurationUnit"
  description="This is a sample configuration unit"
  configurationUnit="true">

  <core:capability
    displayName="SampleDependencyCapability"
    name="sampleDependencyCapability"
    description="This is a sample dependency capability"
    linkType="dependency" />

</core:unit>
```

The same dependency capability added above can also be associated to a SampleUnit, as shown in Listing 14.

Listing 14. Associate the capability to a SampleUnit

```
<core:unit
  displayName="SampleUnit"
  name="sampleUnit"
  description="This is a sample unit">

  <core:capability
    displayName="SampleDependencyCapability"
    name="sampleDependencyCapability"
    description="This is a sample dependency capability"
    linkType="dependency" />

</core:unit>
```

Domain Usage

Specializing capability types by domains provides additional abilities. Typically, a capability is also defined for each of the unit types, such as to contain attributes based on the configuration content of the units in that domain. This facilitates in validating the units dependencies to the specific attributes of the unit. As an example, a J2EE domain may provide for:

- A J2eeServer unit type along with a J2eeContainer that allows you to specify a `containerVersion` attribute
- A ServletContainer capability type that allows you to specify a `containerVersion` attribute
- A J2eeDatasource capability that allows you to specify a `jndiName` attribute
- A J2eeServer capability that may contain configuration attributes like `serverName` and so on

You can then create a topology model that defines an instance of the J2eeServer unit that contains the following, as shown in Listing 15:

- A J2eeContainer hosting capability with a specific `containerVersion` of 1.4
- A ServletContainer hosting capability with a specific `containerVersion` of 2.3
- A J2eeDatsource dependency capability that has a `jndiName` set to `jndi/datasource1`
- A J2eeServer capability that may contain configuration attributes of the J2eeServer

Note that in order to use the types from J2EE domain, a namespace reference is added to the topology file as

```
xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/".
```

Listing 15. Topology model

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
description="This is a sample topology describing the anatomy of the topology model"
displayName="Sample Topology"
name="SampleTopology"
decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
namespace="developerworks.deploy">
<core:unit description="This is a sample unit"
displayName="SampleUnit" name="sampleUnit"/>
<core:unit
description="This is a sample configuration unit"
```

```
displayName="SampleConfigurationUnit"
name="sampleConfigurationUnit"
configurationUnit="true"/>
<core:unit
description="This is a sample conceptual unit"
displayName="SampleConceptualUnit"
name="sampleConceptualUnit"
conceptual="true"
goalInstallState="not_installed"
initInstallState="installed"
publishIntent="do_not_publish"/>
<core:unit
description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit"
name="serviceComponentUnit">
<core:stereotype keyword="service"
profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
name="j2eeServerUnit"
conceptual="true"
goalInstallState="installed"
initInstallState="installed">

<j2ee:capability.j2eeServer

description="This is a J2EE server capability"

displayName="J2EE Server Capability"

name="j2eeServer"

linkType="any"/>

<j2ee:capability.j2eeContainer

description="This is a J2EE container capability"

displayName="J2EE Container Capability"

name="j2eeContainer"

linkType="hosting"

containerVersion="1.4"/>

<j2ee:capability.servletContainer

description="This is a J2EE servlet container capability"

displayName="J2EE Servlet Container Capability"

name="j2eeServletContainer"

linkType="hosting"

containerVersion="2.3"/>

<j2ee:capability.j2eeDatasource

description="This is a J2EE datasource dependency capability"

displayName="J2eeDatasourceCapability"
```

```

name="j2eeDatasourceCapability"
linkType="dependency"
jndiName="jndi/datasource1"/>

</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear
description="This is an EAR Component "
displayName="EAR Component "
name="earComponent"
configurationUnit="false"
goalInstallState="installed"
initInstallState="not_installed"/>
</core:topology>

```

Requirement

Requirement represents the need that a unit may have in the target environment. The requirement type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. Requirement is contained in the unit and is uniquely identified by its name with in the scope of a unit, as shown in Listing 16.

Listing 16. Requirement

```
<core:requirement name="r0"/>
```

Optionally, a `linkType` with a default value of `dependency` is defined on the requirement to specify the type of relationship links that can be used to satisfy the requirement. A `dmoType` attribute on the requirement is used to specify an optional constraint to the target unit or capability type that can be used to fulfill the requirement. You can also specify a display name that is consumable by the visual tools, along with a description. Additionally, a usage of the requirement can also be set as `required`, `optional`, or `prohibited` through the `use` attribute (the default value is `required`).

You can specify an optional dependency requirement that can be satisfied by a J2EE datasource capability using a dependency relationship link, as shown in Listing 18. The `dmoType` attribute value for dependency requirements specifies the constraint to a type of capability.

Listing 18. Specify an optional dependency requirement

```

<core:requirement
displayName="Datasource Dependency Requirement"

```

```
name="r0"
description="This is a sample dependency requirement"
linkType="dependency"
dmoType="j2ee:J2EEDatasource"
use="optional" />
```

You can also specify a required hosting requirement that can be satisfied by a J2EE Enterprise Java™ Beans (EJB) container capability using a hosting relationship, as shown in Listing 19. The `dmoType` attribute value for hosting requirements, specifies the constraint to a type of capability.

Listing 19. Specify a hosting requirement

```
<core:requirement
displayName="EJB Container Hosting Requirement"
name="r1"
description="This is a sample hosting requirement"
linkType="hosting"
dmoType="j2ee:J2eeContainer"/>
```

In addition, you can specify a member requirement on a unit that allows members through the member relationship link, as shown in Listing 20. Also note that in this case, `dmoType` is constrained to a unit type rather than a capability type (as in previous cases).

Listing 20. Specify a member requirement

```
<core:requirement
displayName="Unit Member Requirement"
name="r2"
description="This is a sample member requirement
showing that member of type Unit are allowed where
this requirement is defined "
linkType="member"
dmoType="core:Unit"/>
```

You can also specify the `linkType` as any, but that is rarely used for the capabilities that can simultaneously be `dependency`, `hosting`, and `member`. You can also define an `extends` attribute to specify a relative path to the base requirement in order to create an extension. However, this again is a rare concept

that is used in defining the domains.

Domain Usage

Similar to capabilities, though, requirements are associated with a unit in the topology model to represent the needs of the unit. However, the schema for topology model does not enforce any strict association between unit and requirement types. These are loosely coupled together in the topology models to provide utmost flexibility between the unit and the available requirement types. For instance, the same requirement type can be associated with one or more unit type.

As an example, a topology can be defined such that it contains an instance of a J2EE application component defined in the J2EE domain that has the following, as shown in Listing 21:

- A hosting requirement on J2eeContainer capability
- A dependency requirement on a J2eeDatasource capability
- A member requirement for containing J2EE EJB Component (s)

Listing 21. Topology with J2EE application

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
description="This is a sample topology describing the anatomy
of the topology model" displayName="Sample Topology"
name="SampleTopology"
decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
namespace="developerworks.deploy">
<core:unit description="This is a sample unit"
displayName="SampleUnit" name="sampleUnit"/>
<core:unit description="This is a sample configuration unit"
displayName="SampleConfigurationUnit"
name="sampleConfigurationUnit"
configurationUnit="true"/>
<core:unit description="This is a sample conceptual unit"
displayName="SampleConceptualUnit"
name="sampleConceptualUnit" conceptual="true"
goalInstallState="not_installed"
initInstallState="installed"
publishIntent="do_not_publish"/>
<core:unit description="This is a stereotyped service
component unit" displayName="ServiceComponentUnit"
name="serviceComponentUnit">
<core:stereotype keyword="service"
profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit " name="j2eeServerUnit"
conceptual="true" goalInstallState="installed"
initInstallState="installed">
<j2ee:capability.j2eeServer
description="This is a J2EE server capability"
displayName="J2EE Server Capability"
```

```
name="j2eeServer" linkType="any"/>
<j2ee:capability.j2eeContainer
description="This is a J2EE container capability"
displayName="J2EE Container Capability"
name="j2eeContainer" linkType="hosting"
containerVersion="1.4"/>
<j2ee:capability.servletContainer
description="This is a J2EE servlet container capability"
displayName="J2EE Servlet Container Capability"
name="j2eeServletContainer" linkType="hosting"
containerVersion="2.3"/>
</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear
description="This is an EAR Component "
displayName="EAR Component " name="earComponent"
configurationUnit="false" goalInstallState="installed"
initInstallState="not_installed">

<core:requirement
description="This is a datasource dependency requirement"
displayName="Datasource Dependency Requirement"
name="r0"
dmoType="j2ee:J2EEDatasource"
linkType="dependency"
use="optional"/>

<core:requirement
description="This is a J2EE container hosting requirement"
displayName="J2EE Container Hosting Requirement"
name="r1"
dmoType="j2ee:J2EEContainer"
linkType="hosting"/>

<core:requirement
description="This is a EJB component member
requirement showing that member of type EjbComponent unit are allowed "
displayName="EJB Component Member Requirement"
name="r2"
dmoType="j2ee:EjbModule"
linkType="member"/>

</j2ee:component.ear>
</core:topology>
```

Artifact

An artifact type represents the deployable object that can be contained with a unit. The artifact type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. The unit can contain artifact(s) where each of the artifact instances is uniquely identified by its name within the scope of a unit, as shown in Listing 22. Artifact type is defined as an abstract type in the topology schema. Extensions of this type are usually defined to specify the type of content that an artifact may have. A common type of artifact is a `FileArtifact` that represents a file resource, is defined in the topology schema.

Listing 22. Artifact type

```
<core:artifact.file
  displayName="Sample File Jar "
  name="a1">
  <core:fileURI>samplefile.jar</core:fileURI>
</core:artifact.file>
```

Domain Usage

As an example, you can associate a J2EE application component defined in a topology model with an enterprise archive resource (EAR), as shown in the topology in Listing 23. This resource represents all the deployable content associated with the component that a publisher may deploy when the component is installed on the server.

Listing 23. Associate a J2EE application component with an EAR file

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
  xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
  xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
  description="This is a sample topology describing the anatomy of
  the topology model" displayName="Sample Topology"
  name="SampleTopology"
  decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
  namespace="developerworks.deploy">
  <core:unit description="This is a sample unit"
  displayName="SampleUnit" name="sampleUnit"/>
  <core:unit description="This is a sample configuration unit"
  displayName="SampleConfigurationUnit"
  name="sampleConfigurationUnit"
  configurationUnit="true"/>
  <core:unit description="This is a sample conceptual unit"
  displayName="SampleConceptualUnit"
  name="sampleConceptualUnit"
  conceptual="true" goalInstallState="not_installed"
  initInstallState="installed"
  publishIntent="do_not_publish"/>
  <core:unit
  description="This is a stereotyped service component unit"
```

```

displayName="ServiceComponentUnit"
name="serviceComponentUnit">
<core:stereotype keyword="service"
profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
name="j2eeServerUnit"
conceptual="true"
goalInstallState="installed"
initInstallState="installed">
<j2ee:capability.j2eeServer
description="This is a J2EE server capability"
displayName="J2EE Server Capability"
name="j2eeServer"
linkType="any"/>
<j2ee:capability.j2eeContainer
description="This is a J2EE container capability"
displayName="J2EE Container Capability"
name="j2eeContainer"
linkType="hosting"
containerVersion="1.4"/>
<j2ee:capability.servletContainer
description="This is a J2EE servlet container capability"
displayName="J2EE Servlet Container Capability"
name="j2eeServletContainer"
linkType="hosting"
containerVersion="2.3"/>
</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear
description="This is an EAR Component "
displayName="EAR Component "
name="earComponent"
configurationUnit="false"
goalInstallState="installed"
initInstallState="not_installed">

<core:artifact.file

displayName="EAR Component1 EAR "

name="a1">

<core:fileURI>j2eeEarComponent1.ear</core:fileURI>

</core:artifact.file>

<core:requirement
description="This is a datasource dependency requirement"
displayName="Datasource Dependency Requirement"
name="r0" dmoType="j2ee:J2EEDatasource"
linkType="dependency" use="optional"/>
<core:requirement
description="This is a J2EE container hosting requirement"
displayName="J2EE Container Hosting Requirement"
name="r1" dmoType="j2ee:J2EEContainer"
linkType="hosting"/>
<core:requirement
description="This is a EJB component member requirement showing that
member of type EjbComponent unit are allowed "
displayName="EJB Component Member Requirement"
name="r2" dmoType="j2ee:EjbModule"
linkType="member"/>
</j2ee:component.ear>
</core:topology>

```

Relationship Links

Expressing a typical IT system requires relationships to be expressed between the different parts of the system. These relationships impose some semantics on the system that further assist in validating or understanding the system. The topology model provides a means of further classifying these relationships based on the semantics driving these relationships. It names these relationships when there is an inherent requirement imposed by their semantics. There are different types of relationships that can be expressed on the units contained in a topology, and these are represented as link types. The initial set of such link types include `Dependency`, `Hosting` and `Member Link`. Each of these links hold the source and the target of the link that is participating in that relationship.

Dependency Link

A `DependencyLink` type is used to link a requirement with a link type of `dependency` (or any) to a capability, which indicates that the requirement is fulfilled by the target capability. It also provides a means to enforce compliance of the source requirement against the target capability. `DependencyLink` type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. The requirement defined on the instance of the unit is the container for the dependency link. A `DependencyLink` instance that is defined in the topology model is uniquely identified by its name within the scope of a requirement, as shown in Listing 24.

Listing 24. DependencyLink

```
<core:requirement
name="r1">

<core:link.dependency
name="link1"
source="/Unit1/r1"
target="/Unit2/c1"/>

</core:requirement>
```

Dependency Link Domain Usage

As an example, an instance of a J2EE application component defined in the J2EE domain can have a dependency requirement on a `J2eeDatasource` capability that is provided by the `J2eeDatasourceUnit` in the topology. A dependency link can therefore be created between the component and the datasource unit that holds the source requirement and the target capability, as shown in Listing 25.

Listing 25. Create a dependency link

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
description="This is a sample topology describing the anatomy of
the topology model" displayName="Sample Topology"
name="SampleTopology"
decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
namespace="developerworks.deploy">
<core:unit description="This is a sample unit"
displayName="SampleUnit" name="sampleUnit"/>
<core:unit description="This is a sample configuration unit"
displayName="SampleConfigurationUnit"
name="sampleConfigurationUnit" configurationUnit="true"/>
<core:unit description="This is a sample conceptual unit"
displayName="SampleConceptualUnit" name="sampleConceptualUnit"
conceptual="true" goalInstallState="not_installed"
initInstallState="installed" publishIntent="do_not_publish"/>
<core:unit description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit" name="serviceComponentUnit">
<core:stereotype keyword="service" profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
name="j2eeServerUnit"
conceptual="true" goalInstallState="installed"
initInstallState="installed">
<j2ee:capability.j2eeServer
description="This is a J2EE server capability"
displayName="J2EE Server Capability" name="j2eeServer"
linkType="any"/>
<j2ee:capability.j2eeContainer
description="This is a J2EE container capability"
displayName="J2EE Container Capability" name="j2eeContainer"
linkType="hosting" containerVersion="1.4"/>
<j2ee:capability.servletContainer
description="This is a J2EE servlet container capability"
displayName="J2EE Servlet Container Capability"
name="j2eeServletContainer" linkType="hosting"
containerVersion="2.3"/>
<j2ee:capability.j2eeDatasource
description="This is a J2EE datasource dependency capability"
displayName="J2eeDatasourceCapability"
name="j2eeDatasourceCapability"
linkType="dependency"
jndiName="jndi/datasourcel"/>
</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear description="This is an EAR Component "
displayName="EAR Component "
name="earComponent"
configurationUnit="false" goalInstallState="installed"
initInstallState="not_installed">
<core:artifact.file displayName="EAR Component1 EAR "
name="al">
<core:fileURI>j2eeEarComponent1.ear</core:fileURI>
</core:artifact.file>
<core:requirement
description="This is a datasource dependency requirement"
displayName="Datasource Dependency Requirement"
```

```

name="r0"

dmoType="j2ee:J2EEDatasource" linkType="dependency"
use="optional">

<core:link.dependency
name="r0Toj2eeDatasourceCapability"

source="/earComponent/r0"

target="/j2eeServerUnit/j2eeDatasourceCapability"/>

</core:requirement>
<core:requirement
description="This is a J2EE container hosting requirement"
displayName="J2EE Container Hosting Requirement" name="r1"
dmoType="j2ee:J2EEContainer" linkType="hosting"/>
<core:requirement
description="This is a EJB component member requirement showing that member of
type EjbComponent unit are allowed "
displayName="EJB Component Member Requirement"
name="r2" dmoType="j2ee:EjbModule" linkType="member"/>
</j2ee:component.ear>
</core:topology>

```

Hosting Link

A `HostingLink` type indicates that a unit will be hosted based on the fulfillment of all hosting requirements with hosting capabilities on the target host unit. It basically holds the relationship between a source host unit with a hosting capability and the target hostee unit with a hosting type of `requirement` defined on it. `HostingLink` type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. The (host) unit instance that hosts another (hostee) unit is the container of the hosting link. A `HostingLink` instance defined in the topology model is uniquely identified by its name within the scope of a unit, as shown in Listing 26.

Listing 26. Caption

```

<core:unit
displayName="HostUnit"
name="hostUnit">

<core:link.hosting

name="link1"

source="/hostUnit"

target="/hosteeUnit"/>

</core:unit>

```

Hosting Link Domain Usage

As an example, a topology may be defined so that it contains:

- An instance of a J2eeServer unit that provides J2eeContainer and a ServletContainer capability
- An instance of a HttpServer unit that provides a ServletContainer capability

An instance of a J2EE Application component defined in the J2EE domain can also be defined in the topology such as it contains

- A hosting requirement on a J2eeContainer capability type that is provided by the J2eeServerUnit in the topology
- A hosting requirement on a ServletContainer capability type that is provided by the J2eeServerUnit in the topology

In the above case, a hosting link thus can be created between the J2EE application component (hostee) and the J2eeServer unit (host). However, a valid hosting link cannot be created between the J2EE application component and the HttpServer unit, because all of the hosting requirements cannot be met by the capabilities defined on the HttpServer unit in the topology, as shown in Listing 27.

Listing 27. Valid hosting link

```
<?xml version="1.0" encoding="UTF-8"?>
<core:topology
xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"

xmlns:http="http://www.ibm.com/ccl/soa/deploy/http/1.0.0/"

xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
description="This is a sample topology describing the anatomy of the
topology model"
displayName="Sample Topology" name="SampleTopology"
decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
namespace="developerworks.deploy">
<core:unit description="This is a sample unit"
displayName="SampleUnit" name="sampleUnit"/>
<core:unit description="This is a sample configuration unit"
displayName="SampleConfigurationUnit"
name="sampleConfigurationUnit"
configurationUnit="true"/>
<core:unit description="This is a sample conceptual unit"
displayName="SampleConceptualUnit"
name="sampleConceptualUnit" conceptual="true"
goalInstallState="not_installed"
initInstallState="installed" publishIntent="do_not_publish"/>
<core:unit
description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit" name="serviceComponentUnit">
<core:stereotype keyword="service" profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
```

```

name="j2eeServerUnit"

conceptual="true" goalInstallState="installed"
initInstallState="installed">
<j2ee:capability.j2eeServer
description="This is a J2EE server capability"
displayName="J2EE Server Capability"
name="j2eeServer" linkType="any"/>
<j2ee:capability.j2eeContainer
description="This is a J2EE container capability"
displayName="J2EE Container Capability"

name="j2eeContainer"

linkType="hosting"
containerVersion="1.4"/>
<j2ee:capability.servletContainer
description="This is a J2EE servlet container capability"
displayName="J2EE Servlet Container Capability"

name="j2eeServletContainer"

linkType="hosting"
containerVersion="2.3"/>
<j2ee:capability.j2eeDatasource
description="This is a J2EE datasource dependency capability"
displayName="J2eeDatasourceCapability"
name="j2eeDatasourceCapability" linkType="dependency"
jndiName="jndi/datasource1"/>

<core:link.hosting

name="j2eeServerUnitHostsearComponent "

source="/j2eeServerUnit"

target="/earComponent"/>

</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear description="This is an EAR Component "
displayName="EAR Component "

name="earComponent"

configurationUnit="false" goalInstallState="installed"
initInstallState="not_installed">
<core:artifact.file displayName="EAR Component1 EAR "
name="a1">
<core:fileURI>j2eeEarComponent1.ear</core:fileURI>
</core:artifact.file>
<core:requirement
description="This is a datasource dependency requirement"
displayName="Datasource Dependency Requirement"
name="r0"
dmoType="j2ee:J2EEDatasource" linkType="dependency"
use="optional">
<core:link.dependency name="r0Toj2eeDatasourceCapability"
source="/earComponent/r0" target="/j2eeServerUnit/j2eeDatasourceCapability"/>
</core:requirement>
<core:requirement
description="This is a J2EE container hosting requirement"
displayName="J2EE Container Hosting Requirement" name="r1"

dmoType="j2ee:J2EEContainer"

linkType="hosting"/>

```

```
<core:requirement
description="This is a EJB component member requirement showing that member of
type EjbComponent unit are allowed "
displayName="EJB Component Member Requirement "
name="r2" dmoType="j2ee:EjbModule"
linkType="member"/>
</j2ee:component.ear>

<http:unit.httpServerUnit
displayName="Http Server"
name="httpServerUnit"
conceptual="false">
<http:capability.httpServer
displayName="Http Server Capability "
name="Http Server Capability"
linkType="any"/>
<j2ee:capability.servletContainer
displayName="Servlet Container Capability"
name="servletContainer"
linkType="any"
containerVersion="2.3"/>
</http:unit.httpServerUnit>

</core:topology>
```

Member Link

A `MemberLink` type represents the containment relationship that defines the linkage between two units, where the target of the link is the member and the source is the container. It basically holds the relationship between a source container unit (that has a member type of `requirement` for a specific type of object) and target member unit of that object type. `MemberLink` type is contributed from the <http://www.ibm.com/ccl/soa/deploy/core/1.0.0/> namespace defined in the topology schema. The unit instance that holds the member requirement for other member units is the container of the member link. A `MemberLink` instance defined in the topology model is uniquely identified by its name within the scope of a unit, as shown in Listing 28.

Listing 28. MemberLink instance

```
<core:unit
displayName="ContainerUnit"
name="containerUnit">
```

```

<core:requirement
  displayName="Membership Requirement "
  name="membershipRequirement"
  dmoType="core:Unit"
  linkType="member" />

<core:link.member
  name="link1"

  source="/ContainerUnit"

  target="/MemberUnit1" />

</core:unit>

```

Member Link Domain Usage

As an example, an Enterprise JavaBean module component is linked with a member link to a J2EE application component to describe the module as a member of the application, as shown in Listing 29.

A topology model can be defined such that it contains:

- A J2EE application component that defines a member requirement on an EJB module type of the component.
- An EJB component

In the previous case, a member link can be created so that the source of the link is the J2EE application component, and the target of the link is an EJB component.

Listing 29. EJB component linked to J2EE component

```

<?xml version="1.0" encoding="UTF-8"?>
<core:topology
  xmlns:core="http://www.ibm.com/ccl/soa/deploy/core/1.0.0/"
  xmlns:http="http://www.ibm.com/ccl/soa/deploy/http/1.0.0/"
  xmlns:j2ee="http://www.ibm.com/ccl/soa/deploy/j2ee/1.0.0/"
  description="This is a sample topology describing the anatomy of
  the topology model" displayName="Sample Topology"
  name="SampleTopology"
  decoratorSemantic="com.ibm.ccl.soa.deploy.analysis.ads"
  namespace="developerworks.deploy">
  <core:unit description="This is a sample unit"
  displayName="SampleUnit" name="sampleUnit" />
  <core:unit
  description="This is a sample configuration unit"
  displayName="SampleConfigurationUnit"
  name="sampleConfigurationUnit"
  configurationUnit="true" />
  <core:unit
  description="This is a sample conceptual unit"
  displayName="SampleConceptualUnit"
  name="sampleConceptualUnit"
  conceptual="true"
  goalInstallState="not_installed"
  initInstallState="installed"

```

```

publishIntent="do_not_publish"/>
<core:unit
description="This is a stereotyped service component unit"
displayName="ServiceComponentUnit"
name="serviceComponentUnit">
<core:stereotype keyword="service"
profile="serviceProfile"/>
</core:unit>
<j2ee:unit.j2eeServerUnit
description="This is a J2eeServer unit "
displayName="J2EE Server Unit "
name="j2eeServerUnit" conceptual="true"
goalInstallState="installed"
initInstallState="installed">
<j2ee:capability.j2eeServer
description="This is a J2EE server capability"
displayName="J2EE Server Capability"
name="j2eeServer" linkType="any"/>
<j2ee:capability.j2eeContainer
description="This is a J2EE container capability"
displayName="J2EE Container Capability"
name="j2eeContainer"
linkType="hosting" containerVersion="1.4"/>
<j2ee:capability.servletContainer
description="This is a J2EE servlet container capability"
displayName="J2EE Servlet Container Capability"
name="j2eeServletContainer" linkType="hosting"
containerVersion="2.3"/>
<j2ee:capability.j2eeDatasource
description="This is a J2EE datasource dependency capability"
displayName="J2eeDatasourceCapability"
name="j2eeDatasourceCapability"
linkType="dependency" jndiName="jndi/datasource1"/>
<core:link.hosting name="j2eeServerUnitHostsearComponent"
source="/j2eeServerUnit"
target="/earComponent"/>
</j2ee:unit.j2eeServerUnit>
<j2ee:component.ear
description="This is an EAR Component "
displayName="EAR Component "

name="earComponent"

configurationUnit="false"
goalInstallState="installed"
initInstallState="not_installed">
<core:artifact.file
displayName="EAR Component1 EAR " name="a1">
<core:fileURI>j2eeEarComponent1.ear</core:fileURI>
</core:artifact.file>
<core:requirement
description="This is a datasource dependency requirement"
displayName="Datasource Dependency Requirement"
name="r0" dmoType="j2ee:J2EEDatasource"
linkType="dependency" use="optional">
<core:link.dependency name="r0Toj2eeDatasourceCapability"
source="/earComponent/r0"
target="/j2eeServerUnit/j2eeDatasourceCapability"/>
</core:requirement>
<core:requirement
description="This is a J2EE container hosting requirement"
displayName="J2EE Container Hosting Requirement"
name="r1"
dmoType="j2ee:J2EEContainer"
linkType="hosting"/>
<core:requirement description="This is a EJB component member
requirement showing that member of type EjbComponent unit are allowed "
displayName="EJB Component Member Requirement" name="r2"
dmoType="j2ee:EjbModule" linkType="member"/>

```

```
<core:link.member
name="earComponentContainsejbComponent "
source="/earComponent "
target="/ejbComponent "/>

</j2ee:component.ear>
<http:unit.httpServerUnit displayName="Http Server"
name="httpServerUnit" conceptual="false">
<http:capability.httpServer displayName="Http Server Capability "
name="Http Server Capability" linkType="any"/>
<j2ee:capability.servletContainer
displayName="Servlet Container Capability"
name="servletContainer" linkType="any"
containerVersion="2.3"/>
</http:unit.httpServerUnit>

<j2ee:component.ejb
displayName="EJB Component "
name="ejbComponent "
conceptual="false"
initInstallState="not_installed"
publishIntent="
publish">
<j2ee:capability.ejbModule
displayName="EJB Component "
name="ejbComponentCapability"
linkType="dependency"
id="EJBComponent "
version="1.0"/>
</j2ee:component.ejb>

</core:topology>
```

What you have learned

In this part of the series, you learned about the basic concepts of the topology model used for deployment architecture in Rational Software Architect V7.5. You built a [sample topology](#) that illustrated the different aspects of a topology model, including topology, unit, requirement, capability, and several relationships that can be expressed between them. You also analyzed the design decisions that you should consider when expressing types and relationships of their domain in a topology

model. Stay tuned for the next part in the series, which focuses on some of the advanced concepts of the topology mode:

- Topology abstraction, encapsulation and reuse
- Realization link relationship
- Constraint semantics and usage
- Specialized Constraint Links semantics and usage (e.g. deferred hosting, communication, hosting capacity, collocation, anticolllocation)
- Custom attribute addition
- Using variables in topology

Downloads

Description	Name	Size	Download method
Sample Topology Project used in this article	SampleTopology.zip	2KB	HTTP

[Information about download methods](#)

Resources

Learn

- Understand more about how the new functionality of the UML Modeler component common to both IBM Rational Software Architect Standard Edition Version 7.5 and IBM Rational Software Architect for WebSphere Software Version 7.5 by reading this article [Using the new features of UML Modeler in IBM Rational Software Architect Version 7.5](#).
- Understand more about the new functionality of Rational Software Architect for WebSphere Software Version 7.5 by reading this article [Overview of Rational Software Architect for WebSphere Software Version 7.5](#).
- Visit the [Rational software area on developerWorks](#) for technical resources and best practices for Rational Software Delivery Platform products.
- Subscribe to the [IBM developerWorks newsletter](#), a weekly update on the best of developerWorks tutorials, articles, downloads, community activities, webcasts and events.
- Subscribe to the [developerWorks Rational zone newsletter](#). Keep up with developerWorks Rational content. Every other week, you'll receive updates on the latest technical resources and best practices for the Rational Software Delivery Platform.
- Subscribe to the [Rational Edge newsletter](#) for articles on the concepts behind effective software development.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download [trial versions of IBM Rational software](#).
- Download [Rational Software Architect for Websphere Software 7.5](#).
- Download [Other IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).
- Join the [Rational Software Architect forum](#) on developerWorks.

About the author

Narinder Makin

Narinder Makin is a software architect at IBM Research Triangle Part Lab in Durham, North Carolina. He works in the area of Deployment tools for the Rational Software Architect team. He has worked on a variety of Model Driven Development and Deployment tools over the past 12 years. He has several patents and published articles. He holds an undergraduate degree in Computer Science and Engineering from K.N.I.T, India, and also holds a Masters degree in Computer Science from University Of Bridgeport, Connecticut.

Trademarks

IBM and the IBM logo are trademarks of International Business Machines Corporation in the United States, other countries or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product and service names may be trademarks or service marks of others.